



Programación Python para Big Data

Lección 10: Futuros aprendizajes en Big
Data

ÍNDICE

Lección 10. – Futuros aprendizajes en Big Data	2
Presentación y objetivos	2
1. QISKIT e IBM – Simulación en el Ordenador.....	3
2. QISKIT e IBM – Simulación en Hardware de IBM	12
3. Importancia de las Certificaciones.....	15
4. Certificaciones relacionadas con este Máster	16
5. Otros tipos de Bases de Datos	19
6. Otros campos de aprendizaje en Big Data.....	20
7. Puntos clave.....	21

Lección 10. – Futuros aprendizajes en Big Data

PRESENTACIÓN Y OBJETIVOS

En la presente lección de esta asignatura haremos una presentación de cosas no explicadas en este temario sobre Big Data.

Si bien se ha tratado de condensar el conocimiento que suele hacer falta en estas clases como por ejemplo las herramientas para procesar grandes volúmenes de datos, las Bases de Datos Relacionales y No Relacionales, y las herramientas muy usadas como es el caso de Apache Spark, e incluso una breve introducción a la Computación Cuántica.



Objetivos

- | Conocer certificaciones posibles en Big Data
- | Aprender algo sobre QISKIT e IBM Quantum Computing (programación con Python)
- | Conocer algunas herramientas no vistas en este temario, incluso otras Bases de datos

1. QISKIT E IBM – SIMULACIÓN EN EL ORDENADOR

Instalamos qiskit en cmd

```
In [1]: # !pip install qiskit  
# mejor en CMD previo
```

Importamos todo de qiskit

```
In [2]: # importo todo  
from qiskit import *
```

Figura 1.1: Breve Intro a Qiskit (parte 1)

Algo importante es conocer la versión de qiskit que estamos usando, para ello haremos lo siguiente:

compruebo la versión

```
In [3]: qiskit.__qiskit_version__  
  
c:\users\manut\appdata\local\programs\python\python38\lib\site-packages\qiskit\__init__.py:86: DeprecationWarning: The package qiskit.aqua is deprecated. It was moved/refactored to qiskit-terra. For more information see <https://github.com/Qiskit/qiskit-aqua/blob/main/README.md#migration-guide>  
  warn_package('aqua', 'qiskit-terra')  
  
Out[3]: {'qiskit-terra': '0.18.1', 'qiskit-aer': '0.8.2', 'qiskit-ignis': '0.6.0', 'qiskit-ibmq-provider': '0.16.0', 'qiskit-aqua': '0.9.4', 'qiskit': '0.29.0', 'qiskit-nature': None, 'qiskit-finance': None, 'qiskit-optimization': None, 'qiskit-machine-learning': None}  
  
In [4]: for element, version in qiskit.__qiskit_version__.items():  
        print(element, version)  
  
qiskit-terra 0.18.1  
qiskit-aer 0.8.2  
qiskit-ignis 0.6.0  
qiskit-ibmq-provider 0.16.0  
qiskit-aqua 0.9.4  
qiskit 0.29.0  
qiskit-nature None  
qiskit-finance None  
qiskit-optimization None  
qiskit-machine-learning None
```

Figura 1.2: Breve Intro a Qiskit (parte 2)

```
In [5]: for version in qiskit.__qiskit_version__.items():  
        print(version)  
  
('qiskit-terra', '0.18.1')  
('qiskit-aer', '0.8.2')  
('qiskit-ignis', '0.6.0')  
('qiskit-ibmq-provider', '0.16.0')  
('qiskit-aqua', '0.9.4')  
('qiskit', '0.29.0')  
('qiskit-nature', None)  
('qiskit-finance', None)  
('qiskit-optimization', None)  
('qiskit-machine-learning', None)
```

Figura 1.3: Breve Intro a Qiskit (parte 3)

Creamos el Circuito

La mayoría de los circuitos serán una combinación de registros clásicos y cuánticos

Los registros cuánticos son usados para:

- realizar operaciones mecánicas cuánticas en qubits

Los registros clásicos

- serán usados para realizar operaciones clásicas de las medidas obtenidas

Figura 1.4: Breve Intro a Qiskit (parte 4)

```
In [6]: # registro cuántico de 2 qubits
qr = QuantumRegister(2)
qr

Out[6]: QuantumRegister(2, 'q0')
```

```
In [7]: # Registro clásico de 2 bits
cr = ClassicalRegister(2)
cr

Out[7]: ClassicalRegister(2, 'c0')
```

```
In [8]: # Construimos un circuito con estos 2
circuit = QuantumCircuit(qr,cr)
circuit

Out[8]: <qiskit.circuit.quantumcircuit.QuantumCircuit at 0x1ac5a033fa0>
```

Figura 1.5: Breve Intro a Qiskit (parte 5)

Dibujamos el circuito

```
In [9]: %matplotlib inline

In [10]: circuit.draw()

Out[10]:
q0_0:
q0_1:
c0: 2/
```

Vemos 2 bits cuánticos y 2 bits clásicos

Bits cuánticos (q0_0) y (q0_1)

Figura 1.6: Breve Intro a Qiskit (parte 6)

```
In [11]: # con el objetivo de crear "entanglement" (entrelazamiento cuántico)
# lo 1º es aplicar la puerta Hadamard en el primer qubit (q0_0)
# con ello logramos la "superposición"
# al aplicar el operador H produce que un qubit pase de un estado básico,  $|0\rangle$  o  $|1\rangle$ ,
# a un estado de superposición equiprobable de ambos estados
# no dar mayor importancia ahora a esto..
circuit.h(qr[0])
```

```
Out[11]: <qiskit.circuit.instructionset.InstructionSet at 0x1ac5a029be0>
```

```
In [12]: circuit.draw()
```

```
Out[12]: q0_0: ── H ──
q0_1: ───────────
c0: 2/══════════
```

Figura 1.7: Breve Intro a Qiskit (parte 7)

```
In [13]: # o incluso mejor visualizado..
circuit.draw(output="mpl")
```

```
Out[13]:
```

```
q0_0 ── H ──
q0_1 ───────────
c0 2/══════════
```

Figura 1.8: Breve Intro a Qiskit (parte 8)

Si te sale el circuito impreso 2 veces es un error no resuelto todavía en el Software, no le des mayor importancia.

```
In [14]: # aparece la puerta Hadamard (Hadamard gate)
# tal y como se pudo apreciar

In [15]: # ahora la puerta cx, que es una "controlled-x gate"
# y la puerta hace un NOT en qubit target si el qubit de control esta en estado 1
# por defecto todos los qubits son inicializados a 0

# en la siguiente ecuación:
# el de control (qr[0]),
# el target (qr[1])
```

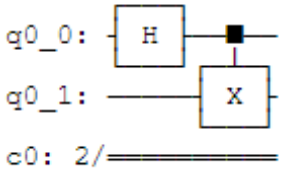
Figura 1.9: Breve Intro a Qiskit (parte 9)

```
In [16]: circuit.cx(qr[0], qr[1])

Out[16]: <qiskit.circuit.instructionset.InstructionSet at 0x1ac5a242b20>

In [17]: circuit.draw()

Out[17]:
```



The diagram shows a quantum circuit with two qubits, q0_0 and q0_1. q0_0 starts with a Hadamard (H) gate. Then, a controlled-X gate is applied, with q0_1 as the control (indicated by a dot) and q0_0 as the target (indicated by a cross). Below the qubit lines, there is a label 'c0: 2/' followed by two parallel horizontal lines.

Figura 1.10: Breve Intro a Qiskit (parte 10)

Hemos hecho el `.draw()` y podemos, de igual manera hacer el gráfico con `output = "mpl"`


```
In [18]: circuit.draw(output="mpl")
```

Out[18]:

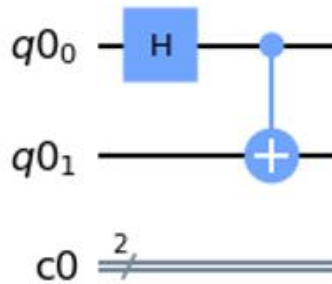


Figura 1.11: Breve Intro a Qiskit (parte 11)

En este caso, de igual manera, si te ha salido impreso 2 veces no le des importancia, ten en cuenta que el resultado está duplicado.

Si te salió bien pues ignora el comentario.

medición de los bits cuánticos

```
In [19]: # un experimento real termina por medir cada qubit.  
# Sin medición no podemos obtener información acerca del estado
```

```
In [20]: circuit.measure(qr,cr)
```

Out[20]: <qiskit.circuit.instructionset.InstructionSet at 0x1ac5a33b4f0>

```
In [21]: circuit.draw()
```

Out[21]:

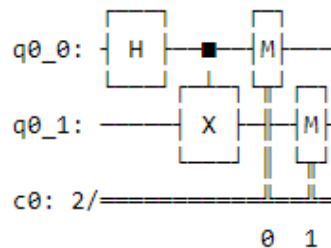


Figura 1.12: Breve Intro a Qiskit (parte 12)

```
In [22]: circuit.draw(output="mpl")
```

Out[22]:

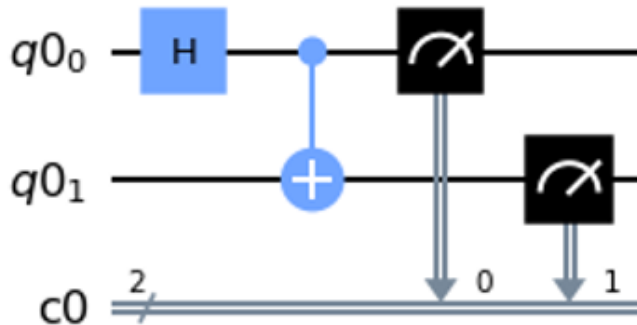


Figura 1.13: Breve Intro a Qiskit (parte 13)

ejecutar el ejercicio en nuestro ordenador

```
In [23]: backend_simulator = Aer.get_backend("qasm_simulator")
         backend_simulator
```

Out[23]: QasmSimulator('qasm_simulator')

```
In [24]: # opción 1
         job_simulator = execute(circuit, backend=backend_simulator)
         job_simulator
         # opción 2 (e incluso..)
         # execute(circuit, backend=simulator, shots=1024)
```

Out[24]: <qiskit.providers.aer.aerjob.AerJob at 0x1ac5a3b1df0>

Figura 1.14: Breve Intro a Qiskit (parte 14)

```
In [25]: result = job_simulator.result()
result

Out[25]: Result(backend_name='qasm_simulator', backend_version='0.8.2', qobj_id='3ef23a80-adf5-438f-92d2-7fcee40a838c', job_id='ba22195b-ecd7-4d0b-894d-852c8fb918cc', success=True, results=[ExperimentResult(shots=1024, success=True, meas_level=2, data=ExperimentResultData(counts={'0x3': 511, '0x0': 513}), header=QobjExperimentHeader(cbit_labels=[['c0', 0], ['c0', 1]], creg_sizes=[['c0', 2]], global_phase=0.0, memory_slots=2, metadata=None, n_qubits=2, name='circuit-9', qreg_sizes=[['q0', 2]], qubit_labels=[['q0', 0], ['q0', 1]]), status=DONE, seed_simulator=1706063825, metadata={'method': 'stabilizer', 'measure_sampling': True, 'parallel_state_update': 4, 'parallel_shots': 1, 'fusion': {'enabled': False}}, time_taken=0.0181189)], date=2021-09-08T14:31:53.776800, status=COMPLETED, status=QobjHeader(backend_name='qasm_simulator', backend_version='0.8.2'), metadata={'parallel_experiments': 1, 'omp_enabled': True, 'max_memory_mb': 15320, 'max_gpu_memory_mb': 0, 'mpi_rank': 0, 'num_mpi_processes': 1, 'time_taken': 0.0186087}, time_taken=0.02199387550354004)

In [26]: result.status

Out[26]: 'COMPLETED'
```

Figura 1.15: Breve Intro a Qiskit (parte 15)

```
In [27]: # plot_histogram
from qiskit.tools.visualization import plot_histogram
```

Figura 1.16: Breve Intro a Qiskit (parte 16)

```
In [28]: plot_histogram(result.get_counts(circuit))
```

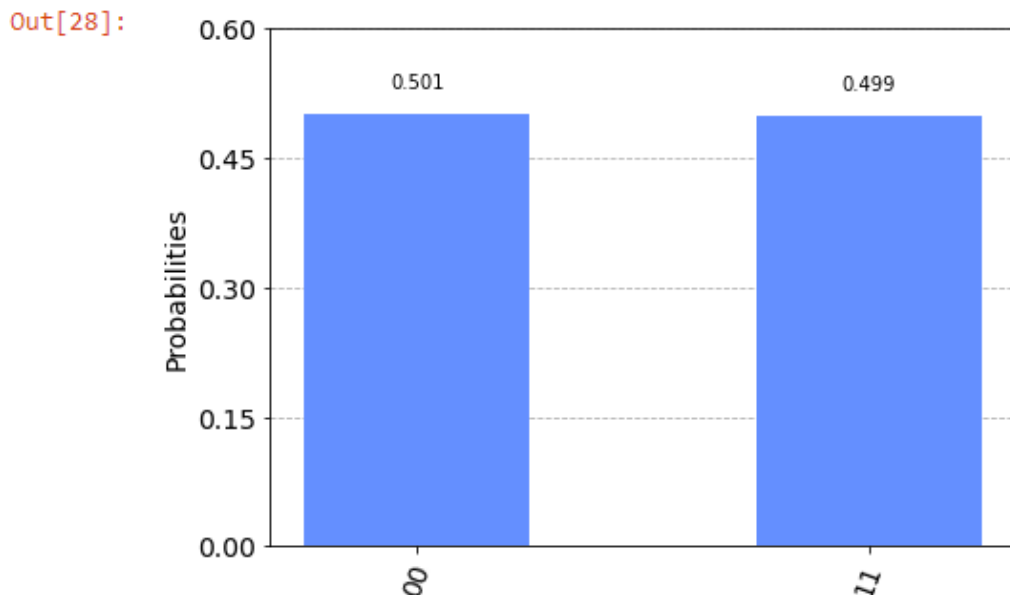


Figura 1.17: Breve Intro a Qiskit (parte 17)

Nuevamente en mi caso fue impreso 2 veces, pero he realizado un solo pantallazo.

Se obtiene cerca de 0.5 en cada uno dado que son un número limitado de “shots”, no infinitos. Tiene por ello una ligera desviación sobre 0.5.

Pero el simulador simula un ordenador cuántico perfecto.

Por el momento trata simplemente de entender en líneas generales la dinámica de trabajo, si algo no lo entiendes perfectamente no te preocupes.

La presente lección tiene como objetivo hacer una pequeña introducción.

Lo próximo que haremos será simular en el hardware de IBM el experimento realizado.

El Hardware de IBM es susceptible, tal y como comentaremos a errores cuánticos. NO LOS DE IBM SOLAMENTE, sino todos a día de hoy, puesto que es un campo en actual investigación actual.

Entonces apreciarás que el resultado no es exactamente el mismo.

No te preocupes, puesto que comentaremos algo al respecto.

Esta parte es muy interesante! Comencemos!

2. QISKIT E IBM – SIMULACIÓN EN HARDWARE DE IBM

Una vez hemos terminado nuestro ejercicio y hemos simulado el mismo en el ordenador, lo que hacemos es irnos a la web de IBM, donde nos hemos registrado en la clase anterior.

Necesitamos un “API TOKEN” que nos proporcionan ellos, tal y como explicamos en la lección anterior nuevamente.

En este caso no hemos añadido en el manual el API TOKEN, por motivos de seguridad. Pero la forma de hacerlo está indicada en la celda 30.

```
IBMQ.save_account("aqui_pegas_tu_API_TOKEN")
```

(El cual te proporcionan ellos cuando te registras, y obviamente es diferente el de cada usuario).

Ejecución en quantum computer de IBM

```
In [29]: from qiskit import IBMQ

In [30]: # IBMQ.save_account('API_TOKEN_COPIADO_DESDE_IBM')
# se ha borrado por seguridad mi API TOKEN
# CADA alumno/a deberá probar con la suya propia.
# Una vez que se ejecuta esta celda, después, se puede omitir, ya se reconoce.
# No debes enseñar ese API TOKEN a los demás

In [31]: IBMQ.load_account()

Out[31]: <AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>

In [32]: IBMQ.providers()

Out[32]: [<AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>]

In [33]: provider = IBMQ.get_provider('ibm-q')
provider

Out[33]: <AccountProvider for IBMQ(hub='ibm-q', group='open', project='main')>
```

Figura 2.1: Ejecución en Hardware de IBM (parte 1)

```
In [34]: provider.backends()

Out[34]: [<IBMQSimulator('ibmq_qasm_simulator') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_armonk') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_santiago') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_bogota') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_lima') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_belem') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_quito') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQSimulator('simulator_statevector') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQSimulator('simulator_mps') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQSimulator('simulator_extended_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQSimulator('simulator_stabilizer') from IBMQ(hub='ibm-q', group='open', project='main')>,
<IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open', project='main')>]
```

Figura 2.2: Ejecución en Hardware de IBM (parte 2)

```
In [35]: # elijo por ejemplo
# <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open', project='main')>]
qcomp = provider.get_backend('ibmq_manila')
qcomp
```

```
Out[35]: <IBMQBackend('ibmq_manila') from IBMQ(hub='ibm-q', group='open', project='main')>
```

```
In [36]: job = execute(circuit, backend=qcomp)
job
```

```
Out[36]: <qiskit.providers.ibmq.job.ibmqjob.IBMQJob at 0x24a6b2e8220>
```

```
In [37]: from qiskit.tools.monitor import job_monitor
```

```
In [38]: job_monitor(job)
```

Job Status: job has successfully run

Figura 2.3: Ejecución en Hardware de IBM (parte 3)

```
In [39]: # una vez dice "Job Status: job has successfully run"
result = job.result()
result

Out[39]: Result(backend_name='ibmq_manila', backend_version='1.0.6', qobj_id='137c50ab-e380-4a83-87ba-5202f77f0d6b', job_id='6138ae6a35e37e6bf22edf5f', success=True, results=[ExperimentResult(shots=1024, success=True, meas_level=2, data=ExperimentResultData(counts={'0x0': 530, '0x1': 14, '0x2': 37, '0x3': 443}), header=QobjExperimentHeader(clbit_labels=[['c0', 0], ['c0', 1]], creg_sizes=[['c0', 2]], global_phase=0.7853981633974483, memory_slots=2, metadata={}, n_qubits=5, name='circuit-9', qreg_sizes=[['q', 5]], qubit_labels=[['q', 0], ['q', 1], ['q', 2], ['q', 3], ['q', 4]]), memory=False)], date=2021-09-08 14:38:17+02:00, status=Successful completion, status=QobjHeader(backend_name='ibmq_manila', backend_version='1.0.6'), time_taken=6.0841522216796875, execution_id='a08b05a2-10a1-11ec-af9f-bc97e15b08d0', client_version={'qiskit': '0.29.0'})]
```

Figura 2.4: Ejecución en Hardware de IBM (parte 4)

```
In [40]: plot_histogram(result.get_counts(circuit))
```

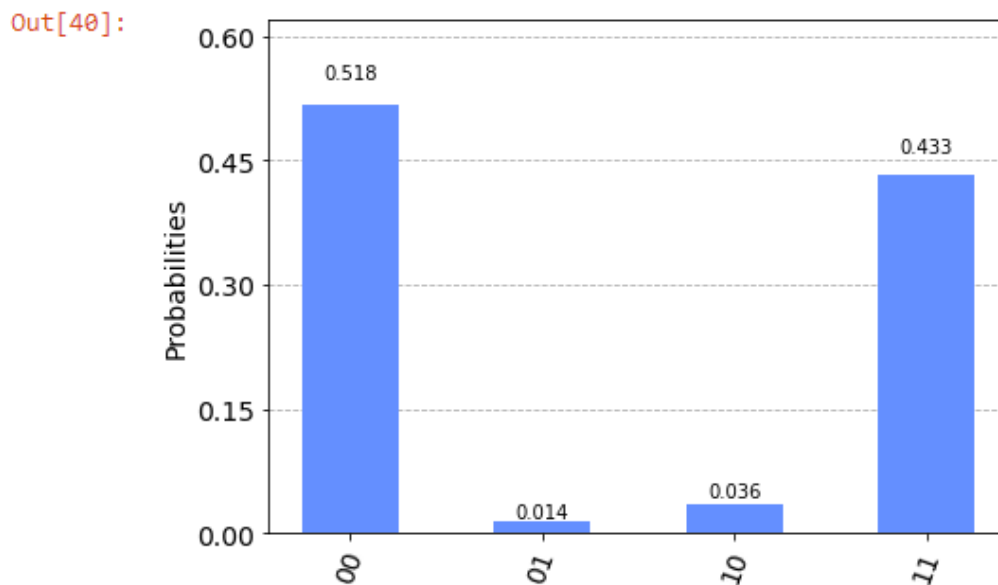


Figura 2.5: Ejecución en Hardware de IBM (parte 5)

Aquí nuevamente me salía la gráfica planteada 2 veces, por lo que si a ti te ocurre lo mismo no lo tengas en cuenta.

Probablemente pronto se corregirán esos errores.

Lo que debe llamarte la atención, y en lo que debes prestar atención es lo explicado en la siguiente celda de código:

```
In [41]: # en este caso tenemos un resultado diferente.
# el simulador lo que hace es simular un ordenador cuántico perfecto.
# pero simulado en un dispositivo cuántico real(de IBM en este caso)
# hay algunos errores, que son corregidos día a día
# (errores cuánticos)
```

Figura 2.6: Ejecución en Hardware de IBM (parte 6)

Y hasta aquí hemos llegado, tal vez puedas aprender más cosas en un futuro, pero por el momento es suficiente.

En este caso trataremos de completar la asignatura con otras partes importantes.

3. IMPORTANCIA DE LAS CERTIFICACIONES

La mayoría de programadores en activo no disponen de certificaciones que avalen sus conocimientos.

Disponer de certificaciones que avalen nuestro conocimiento en algo concreto sería ideal dado que el programa presente de estudios es un programa generalista en Python para 3 ramas concretas.

Pudiéramos especializarnos en cualquiera rama de ellas, incluso en varias al mismo tiempo. Sería posible ser expertos en LINUX sin obtener certificación alguna, de igual manera en TensorFlow, o en Computación Cuántica, etc.

Pero, de alguna manera, aquellos/as que dispongan de certificaciones ello podría suponer un valor diferencial, de igual manera casi que un Máster Universitario.

Vamos a hablar de algunas de las posibles certificaciones que existen en Big Data pero mencionaremos algunas otras relacionadas con el programa de estudios.

Recuerda que hemos hablado de las certificaciones en Tableau, una buena herramienta para Visualización de Datos.

Existen muchas más y de todo tipo.

Podemos continuar hablando sobre ello en el próximo punto (punto 4)

4. CERTIFICACIONES RELACIONADAS CON ESTE MÁSTER

Dentro del propio **Quantum Computing** podemos mencionar la certificación que dispone IBM a fecha de Junio 2021.

| **Quantum Computing: Certificación de Qiskit (IBM)**

<https://www.ibm.com/certify/exam?id=C1000-112>

Ideal para aquellos/as que quieren cualificarse en computación cuántica.

En el futuro probablemente existan certificaciones más avanzadas en este tema, pero es un buen comienzo.

Algunos de los conceptos ya han sido cubiertos en el presente manual.

Existen un montón de certificaciones en el campo de **Cloud Computing**, de hecho existen de diferentes dificultades.

Por citar algunas típicas:

| **Amazon Web Services**

<https://aws.amazon.com/es/certification/>

| **Google Cloud**

<https://cloud.google.com/certification/?hl=es>

| **Azure (Microsoft)**

<https://docs.microsoft.com/es-es/learn/certifications/browse/>

Existen, aunque menos, certificaciones en **Bases de Datos**, destacando la siguiente:

| **Certificación en Bases de Datos MongoDB**

<https://university.mongodb.com/certification/developer/about>

Ya de paso, podemos hablar de certificaciones en **Inteligencia Artificial** y más en concreto Deep Learning, dado que el presente es un Máster que incluye esta temática:

| **Certificación de TensorFlow 2.x (Google)**

<https://www.tensorflow.org/certificate>

Dentro del Deep Learning, disponemos de una certificación muy bien valorada y que puede certificar que dispones de muy buenos conocimientos en la utilización de la herramienta de Google para “Aprendizaje Profundo”.

Sobre esto tal vez tengáis ocasión de aprender algo en las asignaturas específicas de Machine Learning e Inteligencia Artificial

Podemos mencionar alguna cosa como:

- | Se realiza con el Entorno de Desarrollo PyCharm
- | A un precio bastante bajo (unos 100 \$ a fecha Agosto 2021)
- | Necesitas conocer muy bien la herramienta y sus múltiples usos en Procesamiento del Lenguaje Natural (NLP), Clasificación de Imágenes, series temporales, etc.
- | Es una certificación muy reconocida en la industria

Dentro de la **Ciberseguridad**, que también es un tema que de algún modo está relacionado con el Máster académico, hay muchísimas, por destacar una muy común podemos comentar la Certificación de Hacking Ético:

| **“Certified Ethical Hacker (CEH)” .**

<https://www.eccouncil.org/programs/certified-ethical-hacker-ceh/>

En la misma se enseñan las mismas técnicas y las mismas herramientas que pudieran usar los “atacantes”, pero de una forma legal y también controlada.

Es muy popular esta certificación dentro de la ciberseguridad, aunque hay otras muchas más.

5. OTROS TIPOS DE BASES DE DATOS

Para este manual nos hemos decantado por:

- | Bases de Datos Relacionales con PostgreSQL
- | Bases de Datos No Relacionales con MongoDB.

En la mayoría de casos es “suficiente” con conocer esas 2, pero en algunas empresas se emplean otras.

Aunque hay otras, podríamos mencionar estas otras: SQL Server, MySQL, SQLite, etc. Para algunas de ellas indicamos el Link de información

- | **SQLite**

<https://www.sqlite.org/index.html>

- | **Cassandra DB**

https://cassandra.apache.org/_/index.html

Que afirman en su página web que te permiten gestionar ingentes cantidades de datos y de forma muy rápida. (Por ello también útil en Big Data).

Esta tecnología es usada por empresas muy conocidas.

Es del tipo NoSQL

- | **MariaDB**

<https://mariadb.org/>

6. OTROS CAMPOS DE APRENDIZAJE EN BIG DATA

A continuación haremos una breve explicación de aquellas cosas no vistas, o que pudieran verse en mayor profundidad.

No obstante, se ha tratado de cubrir lo esencial sobre Big Data en la presente asignatura. A partir de ahora, será el/la estudiante quién decida qué quiere aprender con mayor interés.

Ecosistema Hadoop

Más cosas del ecosistema Hadoop, Hive etc. En nuestro caso concreto nos hemos decantado por otras tecnologías, (Julio 2021). Pero todavía se siguen usando estas tecnologías.

Más cosas sobre Cloud Computing

Más cosas relacionadas con esto. No solamente la parte comentada de certificaciones. Podría hablarse bastante sobre ello.

| Apache Airflow

<https://airflow.apache.org/>

| Faust Python Stream Processing

<https://faust.readthedocs.io/en/latest/>

<https://faust.readthedocs.io/en/latest/introduction.html>

Una librería para hacer aplicación con data streaming en Python

Y, con el tiempo, probablemente saldrán nuevas y mejores herramientas, de hecho, no nos ha quedado tiempo para hablar de Cirq de Google.

7. PUNTOS CLAVE

- | Existen muchas certificaciones ideales para la demostración de conocimientos en Ciencia de Datos y Big Data.
- | IBM dispone de su propia certificación en Computación Cuántica, y es Python el Lenguaje de Programación que necesitamos.
- | Una tendencia en Big Data actualmente es Data streaming
- | El Big Data es un campo en constante mejora, y en el cual saldrán con los años nuevas tecnologías mejores que las actuales.

